Contents

1 Module `Urm` : Unlimited register machines **1**

1 Module `Urm` : Unlimited register machines

This module provides various types and functions for manipulating unlimited register machines (URMs). This implementation is besed on the specification provided in Ramin Naimi's URM simulator, which is in turn based on that of *Computability, An introduction to recursive function theory* by Nigel J. Cutland. For example, the instruction set is given by:

- `T(n, m)` transfers the contents of the `n`-th register to `m`-th register.
- `Z(n)` sets the value of the `n`-th register to zero.
- `S(n)` increments the value of the `n`-th register.
- `P(n)` decrements the value of the `n`-th register.
- `J(n, m, i)` jumps to the `i`-th instruction if the values of the `n`-th and `m`-th coincide. The instruction count starts at 1, so `i = 1` jumps to the first instruction.

The register set, however, is a bit different. In Nigel's specification the registers are indexed by natural numbers (starting from 1), while in this implementation the registers are indexed by arbitrary integers. Also, in Nigel's specification registers store natural numbers (starting from 0) and in this implementation they store arbitrary integers. I have also added the `P` instruction to help users handle negative integers.

See also

- URM simulator.[`sites.oxy.edu/rnaimi/home/URMsim.htm`]
- Computability, An introduction to recursive function theory.[`doi.org/10.1017/CBO9781139171496`]

```
type instruction =
  | T of int * int
  | Z of int
  | S of int
  | P of int
  | J of int * int * int
```

The type of instructions.

```
type t
```

The type of all possible states of a URM.

```
type machine = t
```

An alias for `Urm.t`[1]

`val of_registers : (int -> int) -> machine`

Returns a machine whose register values are given by a function.

`val of_list : (int * int) list -> machine`

Returns a machine whose registers are given by the pairs in a list. All other registers are zero initialized.

`val register : machine -> int -> int`

`register m i` returns the value of the `i`-th register of `m`.

`val zeros : machine`

A machine whose registers are all zeros.

`val exec : instruction array -> machine -> machine`

`exec m program` returns a machine whose register values are given by the register values of `m` with updates corresponding to the execution of `program`.

This function only returns when `program` actually halts. Please use `Urm.nexec`[1] if you don't trust `program`.

Raises `Invalid_argument` if `program` attempts to jump to a non-positive integer.

`val nexec : int -> instruction array -> machine -> machine option`

A safe version of `Urm.exec`[1], where the maximum number of clock-cycles allowed is controlled the first argument. Each instruction takes a single clock-cycle to execute.

`nexec n program` returns `Some _` if `program` halts in at most `n` clock-cycles and `None` otherwise.

Raises

- `Invalid_argument` if `program` attempts to jump to a non-positive integer.
- `Invalid_argument` if `n` is not a positive integer.

`exception Syntax_error of string`

Exception raised when parsing fails.

`val parse : string -> instruction array`

Parse a program from a string. The syntax used by the parser is based on the one used by Ramin Naimi's URM simulator, but again the register indexes can be arbitrary integers.

Raises `Syntax_error` in case any syntax error is encountered, including invalid syntax and jumps to invalid addresses.

See also URM simulator.[`sites.oxy.edu/rnaimi/home/URMsim.htm`]